STAT 672 Final Project: Stochastic Gradient Descent

Tom Wallace

April 15, 2018

1 Introduction

1.1 Organization

This paper is divided into four sections. The remainder of this **Introduction** section gives intuitive motivation for stochastic gradient descent (SGD). The **Method and Theory** section more rigorously presents the mathematics of SGD and some of its notable properties. The **Applications** sections highlights the real-world settings and uses of SGD, including a case study data analysis. The **Conclusion** section summarizes overall findings.

1.2 Motivation

Optimization is fundamental to statistical modeling. The chief task of statistical modeling is to characterize the relationship between explanatory variables and an outcome variable, and the chief method for doing so is to estimate values for coefficients that best relate each explanatory variable to the outcome variable. The term "best" implies picking coefficient values that maximize some measure of goodness (e.g. likelihood) or minimize some measure of badness (e.g. loss function). Mathematical optimization is the typical route to achieving such minimization or maximization. Two important considerations for optimization are parametric assumptions and computational complexity. SGD, an optimization technique, is particularly motivated by these considerations.

1.2.1 Parametric vs. non-parametric

Assuming that the outcome variable follows a particular statistical distribution aids the computation of optimal coefficients. For example, assumptions in ordinary least squares (OLS) regression—assumptions that readers almost certainly are familiar with and so will not be repeated here—allow a closed form solution. The optimization problem of choosing coefficient $\hat{\boldsymbol{\beta}}$ that minimize squared error is solved by $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$ (where $\boldsymbol{X}$ is the feature matrix and $\boldsymbol{Y}$ the outcome variable).

Even if a parametric model does not have a closed-form solution, the parametric assumption allows some useful optimization techniques. Consider logistic regression. The maximum likelihood estimator (MLE) approach for estimating coefficients leads to a system of D equations. This system of equations typically is numerically solved using the iterative Newton-Raphson algorithm:

$$\hat{\boldsymbol{\beta}}_{n+1} = \hat{\boldsymbol{\beta}}_n - \boldsymbol{H}^{-1}(\hat{\boldsymbol{\beta}}_n)\boldsymbol{J}(\hat{\boldsymbol{\beta}}_n)$$

$\boldsymbol{J}$ is the Jacobian (the first derivative of the log-likelihood function l with respect to each w_j) and $\boldsymbol{H}$ is the Hessian (the second derivative of l with respect to $w_j, w_{j'}$). The practicality of Newton-Raphson thus depends on whether it is convenient to find $\boldsymbol{J}$ and $\boldsymbol{H}$. It is convenient for logistic regression because parametric and independent-and-identically-distributed (IID) assumptions mean l is a simple sum of the log probability distribution function (PDF, in this case binomial) for each observation. We "know" (assume) the form of this PDF and so are confident that the second derivative exists and is not too onerous to calculate. In non-parametric settings, we often cannot be so certain and face the possibility of $\boldsymbol{H}$ being non-existent or cumbersome.

The need to conduct optimization in non-parametric settings is a chief motivation for gradient descent (GD), of which SGD is a variant. In non-parametric settings—most notably supervised and unsupervised statistical learning, in which we again seek to find optimal coefficients to relate input variables to output variables for the purposes of classification or regression—there typically is no closed form solution for the coefficients. It also may not be convenient to find and evaluate the Hessian, making Newton-Raphson undesirable. SGD does not require any parametric assumptions. In its most basic form, SGD only requires finding the gradient (though some extensions do need the Hessian or an approximation to it). SGD thus is well-suited for non-parametric settings.

1.2.2 Computational Complexity

How an optimization technique scales with sample size n is another important consideration. It is little comfort if a method reaches the correct solution but requires an excessive amount of time to do so. "Plain" or "batch" GD requires evaluating the gradient for every single observation, every single iteration, until the algorithm converges. For example, for a dataset of $n = 10^6$ that required 25 iterations to converge, batch GD would require evaluating the gradient 25×10^6 times. This scaling with n can cause untenably long computation time.

SGD alleviates these computational difficulties by requiring the gradient to be evaluated for only a single randomly chosen observation per iteration. This approach means convergence is "noisier" and hence requires more iterations to converge, but each iteration is less complex to compute and so can be done faster. SGD thus scales much more favorably with n than GD, and so is particularly useful for large-n applications such as machine learning and big data problems.

2 Method and Theory

2.1 Basic Form

This section presents the basic form of GD and SGD to set up a more detailed examination in subsequent sections. Suppose we want to minimize a function $f : \mathbb{R}^D \to \mathbb{R}$.

$$\min_w f(w) \tag{1}$$

f takes as input $w \in \mathbb{R}^D$. Assume that f is convex and differentiable, i.e. we can compute its gradient with respect to w, $\nabla f(w)$. The optimal w, i.e. that which minimizes (1), is denoted $\hat{w}$. The iterative GD algorithm for finding $\hat{w}$ is:

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}) \tag{2}$$

t refers to a particular iteration. Assume that we have supplied an initial starting guess for w for $t = 0$, $w^{(0)}$. γ refers to step size (also called learning rate). Assume for now that γ is fixed. The GD algorithm iterates until some stop condition is met. This may be a fixed number of iterations, or that the quality of approximation meets some predefined threshold. A common stopping condition is when the L2 norm of the gradient is less than some arbitrarily small constant.

$$||\nabla f(w^{(t)})||_2 \leq \epsilon \tag{3}$$

Consider a modified situation. Suppose f now takes two arguments, w and $X \in \mathbb{R}^D$. $f : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. We have n observations of X, and denote by $\boldsymbol{X}_{n \times D}$ the matrix of these observations, with X_i being a particular observation or row in that matrix. We apply f over all observations, i.e. $\frac{1}{n} \sum_{i=1}^n f(w, X_i)$. Our new problem is:

$$\min_w \frac{1}{n} \sum_{i=1}^n f(w, X_i) \tag{4}$$

We could apply the GD algorithm above to find the optimal w.

$$w^{(t+1)} = w^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^n \nabla f(w^{(t)}, X_i) \tag{5}$$

But, note that doing so requires evaluating the gradient at every single observation $i \leq n$. This may be computationally intractable for large n and high-dimensional D. The innovation of SGD is to instead evaluate only a single randomly-chosen i at each iteration t.

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}, X_i) \tag{6}$$

As it turns out, SGD converges to $\hat{w}$ and does so in a computationally advantageous way compared to GD.

2.2 Key Properties

2.2.1 Correctness

This section gives both intuition and (partial) proof for why (S)GD converges to $\hat{w}$.

Intuitively, our assumption that f is convex means that there is one critical point and it is the global minimum. Basic calculus tells us that this critical point is located where $f' = 0$ (in one dimension) or $\nabla f = 0$ (in higher dimensions). So our task is to search in the space of f for that point. We start with some initial guess $w^{(0)}$, and every iteration, move "down" the gradient in search of zero. Every iteration, we check if the gradient is equal to 0; if not, we keep moving "down." If the gradient is arbitrarily close to zero, then we have found the critical point, which must be the value of w that minimizes f and hence is $\hat{w}$ (or at least a close approximation to it).

For a more formal proof—following that of Tibshirani 2013—assume (in addition to previously stated assumptions that f is convex and differentiable) that the gradient of f is Lipschitz-continuous with constant L, i.e. that $||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2$ for any x, y. This implies that $\nabla^2 f(w) - LI$ is a negative semi-definite matrix. We perform a quadratic expansion of f around $f(x)$ and obtain the following inequality:

$$\begin{aligned} f(y) &\leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}\nabla^2 f(x)||y - x||_2^2 \\ &\leq f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}L||y - x||_2^2 \end{aligned} \tag{7}$$

Now, suppose we use the GD algorithm presented in (2) with $\gamma \leq \frac{1}{L}$. Denote $x^+ = x - \gamma\nabla f(x)$ and substitute x^+ in for y:

$$\begin{aligned} f(x^+) &\leq f(x) + \nabla f(x)^T(x^+ - x) + \frac{1}{2}L||x^+ - x||_2^2 \\ &= f(x) + \nabla f(x)^T(x - \gamma\nabla f(x) - x) + \frac{1}{2}L||x - \gamma\nabla f(x) - x||_2^2 \\ &= f(x) - \nabla f(x)^T\gamma\nabla f(x) + \frac{1}{2}L||\gamma\nabla f(x)||_2^2 \\ &= f(x) - \gamma||\nabla f(x)||_2^2 + \frac{1}{2}L\gamma^2||\nabla f(x)||_2^2 \\ &= f(x) - (1 - \frac{1}{2}L\gamma)\gamma||\nabla f(x)||_2^2 \end{aligned} \tag{8}$$

We have defined $\gamma \leq \frac{1}{L}$. This implies:

$$-(1 - \frac{1}{2}L\gamma) = \frac{1}{2}L\gamma - 1 \leq \frac{1}{2}L\frac{1}{L} - 1 = \frac{1}{2} - 1 = -\frac{1}{2}$$

Returning to (8), we obtain:

$$f(x^+) \leq f(x) - \frac{1}{2}\gamma||\nabla f(x)||_2^2 \tag{9}$$

By definition, a squared L2 norm will always be positive unless its content is 0, and we have defined γ to be positive, so $\frac{1}{2}\gamma||\nabla f(x)||_2^2$ will always be positive unless the gradient is equal to zero. So, (9) implies that GD results in a strictly decreasing objective function value until it reaches the point where the gradient equals zero, which is the optimal value. A key caveat is an appropriately chosen γ, a point covered in more detail later.

The above proof is for GD. A rigorous proof is not given for why SGD also converges to the optimal value. Informally, we can note that the above proof says given infinite t and appropriate γ, the iterative algorithm will always converge to the optimal value. SGD is doing the same thing as GD, just with a single observation per iteration rather than the entire dataset per iteration. Since SGD is using less information per iteration, we would expect the convergence to require more iterations. But per (9) it is guaranteed to *eventually* arrive at the optimal solution. Put differently, the difference between GD and SGD must then primarily revolve around convergence speed, not the final value that is converged to.

2.2.2 Speed

Table 1—a simplified version of that in Bottou 2010—illustrates the computational advantages of SGD over GD. As shown in row 1, because SGD uses less information per iteration than GD, it requires more iterations to achieve some fixed degree of accuracy ρ. However, row 2 shows that because GD computes the gradient for every observation every iteration, while SGD only does so for a single randomly chosen iteration, GD's time per iteration scales linearly with n while SGD's is a constant. Thus, we conclude that SGD's time to reach accuracy ρ scales only with ρ, while GD's scales with both ρ and—crucially—linearly with n. So, the larger n grows, the larger SGD's computational advantage over GD.

Table 1: Asymptotic comparison of GD and SGD

	GD	**SGD**
Iterations to accuracy ρ	$\log \frac{1}{\rho}$	$\frac{1}{\rho}$
Time per iteration	n	1
Time to accuracy ρ	$n \log \frac{1}{p}$	$\frac{1}{\rho}$

We can visually depict SGD's "noisier" convergence in comparison to GD.

2.3 Extensions

The basic SGD algorithm has been extended in many different ways. The popularity of the algorithm disallows a comprehensive or detailed treatment of all

development. This sub-section covers some of the more important and interesting extensions.

2.3.1 Step Size

As hinted at in our proof of convergence to $\hat{w}$, much depends on an appropriately chosen step size γ. If γ is too small, (S)GD may take an excessively long time to run (since we are only moving a small distance "down" the gradient every iteration); if γ is too large, then an iteration's step may "overshoot" the critical point and prevent convergence. An entire literature has developed around the best way to select γ. The central insight that is common to most γ-selection methods is to treat it as a time-indexed rather than fixed hyperparameter, i.e., $\gamma^{(t)}$ rather than γ. Ideally we would like $\gamma^{(t)}$ to be large when far away from the critical point (so as to increase speed of convergence) and to be small when close to the critical point (so as to avoid overshooting).

Line search is one method for computing step size. As described in Boyd and Vandenberghe 2004, there are two main variants: *exact* and *backtracking*. In exact line search, γ is chosen to minimize f along the ray $\{x + \gamma\Delta x\}$, where $\gamma \in \mathbb{R}^+$) and Δx is the descent direction determined by SGD:

$$\gamma = \underset{s \geq 0}{\operatorname{argmin}} f(x + s\Delta x) \tag{10}$$

This obviously is an optimization problem in and of itself, and so it can be computationally impractical to add this burden in addition to the "main" optimization problem we are trying to solve using SGD. For this reason, *backtracking* is an iterative approximation method that is computationally lighter. The backtracking algorithm takes two hyper-parameters $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$. It starts with an initial guess of $s^{(0)} = 1$, and then for every iteration t, updates $s := \beta t$. The algorithm stops when $f(x + s\Delta x) \leq f(x) + \alpha s \Delta f(x)^T \Delta x$, and the final value of s is taken as γ.

Bottou 2012 advocates using learning rates of the form $\gamma^{(t)} = \gamma^{(0)}(1 + \gamma^{(0)}\lambda t)^{-1}$. When the Hessian H of f is strictly positive, using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$, where $\lambda_{\min}$ is the smallest eigenvalue of H, produces the best convergence speed. Note that $(\lambda_{\min} t)^{-1}$ decreases asymptotically with t, matching our intuition about larger step sizes in early iterations and smaller step sizes in later iterations. However, simply using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$ can produce *too* large of steps in early iterations. Hence, it often works better to start with some reasonable initial estimate $\gamma^{(0)}$ that then decays in the fashion of $(\lambda_{\min} t)^{-1}$, leading to the expression in the first sentence of this paragraph. By definition, "reasonable" is more of a judgment call than a mathematical statement: Bottou provides some heuristics for creating such an estimate of $\gamma^{(0)}$.

2.3.2 Momentum and Acceleration

In our set-up of SGD, we stipulated that the function to be minimized, f, is convex. Although this is a necessary condition for some proofs of SGD's correctness, SGD is widely used in applications—most prominently, neural networks

and deep learning—where this assumption is not always true (as covered in more detail in CHAPTER X). In such settings, it is possible that there are multiple local minima. The challenge for SGD is how to avoid getting "stuck" in these local ravines and continue iterating until the true global minimum is achieved.

Momentum—also called acceleration—is a common technique for avoiding this. As outlined in Rumelhart, Hinton, McClelland, et al. 1986 and Qian 1999, we add a new acceleration term to the familiar GD equation. Let $z^{(t+1)} = \alpha z^{(t)} + \nabla f(w^{(t)})$. GD with momentum then is:

$$w^{(t+1)} = w^{(t)} - \gamma z^{(t+1)} \tag{11}$$

Now, the weight vector obtained for the current timestep depends both on the current gradient *and the gradient of the previous time-step.* Intuitively, without acceleration, SGD tends to oscillate within a ravine because the gradient is dominated by the steep curves on either side of the ravine; acceleration helps point the descent along the shallower curve leading out of the ravine. Nesterov 1983 provides another popular implementation of acceleration. Polyak and Juditsky 1992

2.3.3 Averaging

2.3.4 Predictive Variance Reduction

2.3.5 Parallelization

SGD is commonly used in large-n, computationally demanding applications. Thus, even though SGD is a computational improvement over batch GD, there has been interest in whether SGD can be made even faster by parallelizing it. Zinkevich et al. 2010 present novel algorithms for doing so. The actual algorithms are strikingly simple; their proof is highly technical and omitted here.

The parallelization technique essentially is averaging. In line with previous notation, suppose we have fixed learning rate γ,

Advantages

3 Applications

Shalev-Shwartz et al. 2011

3.1 SGD and Statistical Learning

Dal Pozzolo et al. 2015

Loosely following the set-up of , consider a typical supervised classification problem. We have feature matrix $\boldsymbol{X}_{n\times D, n\in\mathbb{R}^n, D\in\mathbb{R}^D}$ (corresponding to n observations and D features) and labels $\boldsymbol{Y}_{n\times 1}, Y_i \in \{-1, 1\}$. The goal is to predict a particular observation's label Y_i using that observation's features $\boldsymbol{X}_i$. We have a hypothesis class $\mathcal{F}$ consisting of various functions $f_{\boldsymbol{w}}(\boldsymbol{X}_i) \in \mathcal{F}$ parametrized

by weight vector $\boldsymbol{w}$. We have loss function $L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i))$ that expresses the cost of mis-classification. Assume for now that L is convex. We will consider the optimal function $\hat{f}_{\boldsymbol{w}}(\boldsymbol{X}_i) \in F$ that which minimizes empirical risk over all observations: $\frac{1}{n}\sum_{i=1}^{n} L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i))$. Denote $\hat{\boldsymbol{w}}$ the weight coefficients of this optimal function. We thus have:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f_{\boldsymbol{w}}(\boldsymbol{X}_i)) \tag{12}$$

Because L is convex, there is one critical point, it is the global minimum, and it is located where the gradient of the loss function with respect to $\boldsymbol{w}$ is zero. GD is an iterative algorithm that can be used to numerically approximate this point. We first state it in a general form, and then adapt it to the specific problem outlined above.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \frac{1}{n}\sum_{i=1}^{n} \nabla L(Y_i, f_{\boldsymbol{w}_t}(\boldsymbol{X}_i)) \tag{13}$$

t refers to iterations. γ is a parameter controlling the step size (also called "learning rate"). Assume for now that γ is fixed. For $t = 0$ we must supply an initial starting guess, $\boldsymbol{w}_0$.

$$\left| \frac{1}{n}\sum_{i=1}^{n} \nabla L(\boldsymbol{Y}_i, f_{\boldsymbol{w}_t}(\boldsymbol{X}_i)) \right| \leq \epsilon \tag{14}$$

Note that the batch GD algorithm presented in (2) requires evaluating the gradient for every single observation i. In a large-n dataset, this can be computationally infeasible. SGD's innovation is to instead evaluate only a single randomly chosen observation i at each iteration t.

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla L(Y_i, f_{\boldsymbol{w}_t}(\boldsymbol{X}_i)) \tag{15}$$

4 Conclusion

References

[1] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[2] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] Andrea Dal Pozzolo et al. "Calibrating probability with undersampling for unbalanced classification". In: *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE. 2015, pp. 159–166.

[5] Yurii Nesterov. “A method of solving a convex programming problem with convergence rate O(1/sqr(k))”. In: *Soviet Mathematics Doklady* 27 (1983), pp. 372–376.

[6] Boris T Polyak and Anatoli B Juditsky. “Acceleration of stochastic approximation by averaging”. In: *SIAM Journal on Control and Optimization* 30.4 (1992), pp. 838–855.

[7] Ning Qian. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1 (1999), pp. 145–151.

[8] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. “A general framework for parallel distributed processing”. In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1986), pp. 45–76.

[9] Shai Shalev-Shwartz et al. “Pegasos: Primal estimated sub-gradient solver for svm”. In: *Mathematical programming* 127.1 (2011), pp. 3–30.

[10] Ryan Tibshirani. *EECS 10-725: Optimization, Lecture 6*. Sept. 2013.

[11] Martin Zinkevich et al. “Parallelized stochastic gradient descent”. In: *Advances in neural information processing systems*. 2010, pp. 2595–2603.